ONR42
LTR92-016

(1)

FINAL REPORT

A HIGH FIDELITY SIMULATION OF OPTICAL DETECTION SYSTEMS

Volume No. 2

By:

N. R. Guivens, Jr. and P. D. Henshaw

Contract No N00014-90-C-0199

November 30, 1992

SPARTA, Inc.
24 Hartwell Avenue
Lexington, MA 02173

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	NOVEMBER 30, 1992	FINAL REPORT 8/14/91 - 11/30/92

4. TITLE AND SUBTITLE

A HIGH FIDELITY SIMULATION OF OPTICAL DETECTION SYSTEMS

VOLUME 2

5. FUNDING NUMBERS

6. AUTHOR(S)

N.R. GUIVENS, JR. AND P.D. HENSHAW

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SPARTA, INC.
24 HARTWELL AVENUE
LEXINGTON, MA 02173

8. PERFORMING ORGANIZATION REPORT NUMBER

LTR92-016

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

OFFICE OF NAVAL RESEARCH
800 N. QUINCY STREET
ARLINGTON, VA 22217-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

N00014-90-C-0199

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes continued development of a sensor simulation code for high fidelity simulation of active and passive optical detection systems. The primary goal of this work has been to develop a toolbox of procedures which can be combined arbitrarily to model a wide variety of optical detector systems at the component, subsystem, and system level.

the simulation is text file (script) driven, allowing libraries of detectors and detection systems to be assembled. The lowest level routines are primarily mathematical operations, and thus other systems, such as sensor signal processing, can also be modeled.

The report provides an overview and results of model operation. Detailed descriptions of the noise model, the syntax of the modeling language, the available mathematical routines, adn the prototype code structure are provided.

14. SUBJECT TERMS	15. NUMBER OF PAGES
optical detection, sensor design, sensor simulation, sensor system analysis, sensor signal processing	43
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

Table of Contents

A High Fidelity Simulation of Optical Detection Systems

1. Introduction

This study, entitled "Program Definition and System Evaluation for a Short Wavelength Laser Radar," consisted of four tasks: define system needs for short wavelength laser radars, simulate potential system performance, analyze the effect of key system parameters, and simulate and evaluate the performance of potential system demonstrations.

Early in the study, the team determined that detector performance will be critical for the operation of short wavelength laser radar systems. Most of these systems will use direct detection, so quantum efficiency and detector noise performance will directly affect the size and weight of the system. Poor detector performance will require either larger lasers for illumination or larger receiver apertures for a given level of system performance, and both of these system elements are major contributors to the overall system weight. The detector models currently available in the the Defense Laser/Target Signatures (DELTAS) code do not permit high fidelity evaluation of the effect of specific detector subsystems on the overall system performance. The authors filled this gap by developing a high fidelity simulation of optical detection systems that is compatible with both the DELTAS code and SPARTA's optical sensor simulation, SENSORSIM, that simulate laser radar systems designed for a wide variety of missions. This volume describes the result of this simulation development effort.

Sensor simulation codes like DELTAS and SENSORSIM, illustrated in Figure 1, synthesize representative examples of signatures recorded by actual sensor systems that are useful for many applications [1] including sensor design and analysis and development of signal and image processing systems. Prior studies [2] have demonstrated that computer simulation is an efficient and economical tool for assessing the effect of detection systems on the performance of an optical sensor system, but these studies also identified certain inadequacies in existing models of detection systems.

An optical detection system may range from a single device such as a CCD array or a photomultiplier tube to a relatively complex system like the example in Figure 2. This figure shows an image intensifier tube containing a photocathode, a microchannel plate intensifier, and a phosphor on the face of an optical fiber bundle that couples the tube to a CCD array. Many existing detector models constrain input to a precise set of parameter values describing the detection systems under consideration. This approach leads to a dilemma. Models with only a few parameters, usually describing only a simple detector, cannot simulate complex systems like the example in Figure 2 to adequate fidelity. On the other hand, models with enough parameters to simulate fairly complex detection systems are too unwieldy for use with simple detection systems. Additional limitations can arise if the set of parameters chosen for a particular model does not provide the flexibility to simulate some types of devices. A useful general model must allow the user's input to conform to the design and requirements of the detection system.

The authors have developed a robust model that can simulate any type of optical detection system [3]. This model is fully compatible with the SENSORSIM and DELTAS

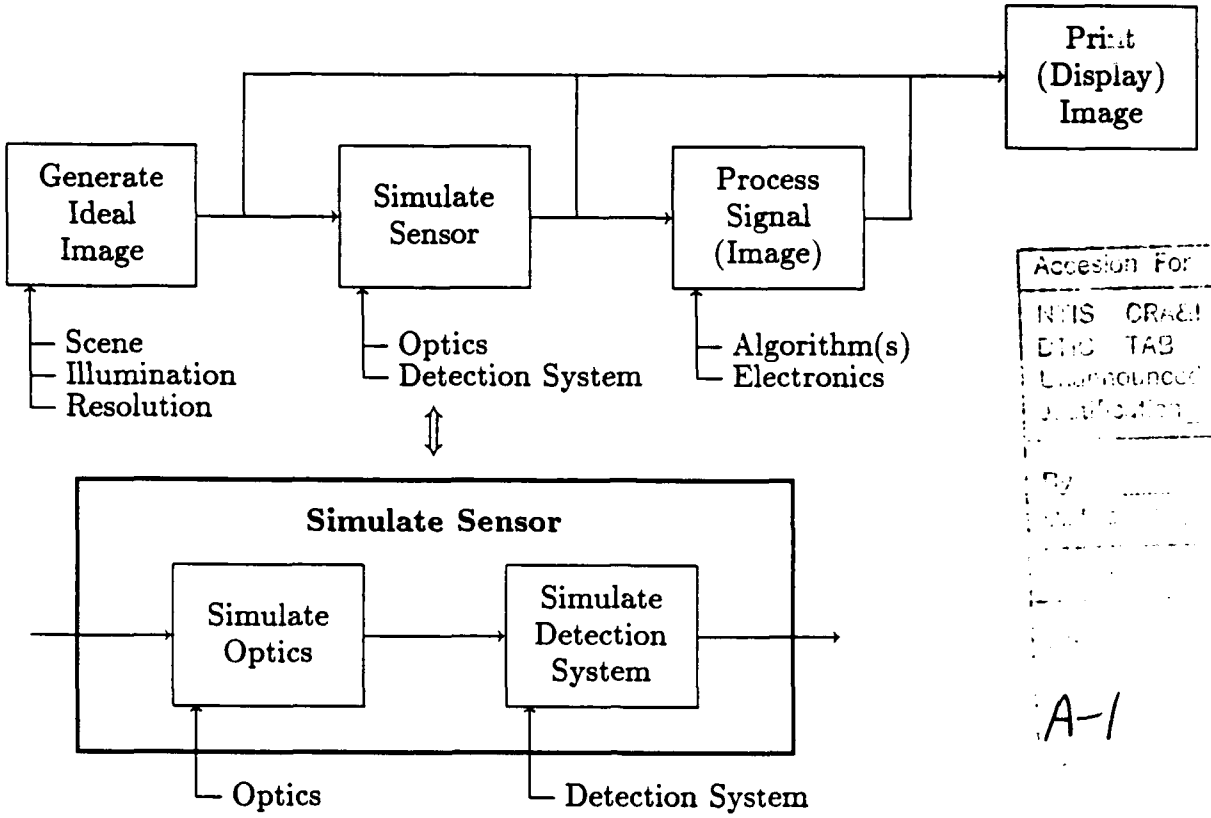


Figure 1. Structure of SENSORSIM, SPARTA's optical sensor simulation.

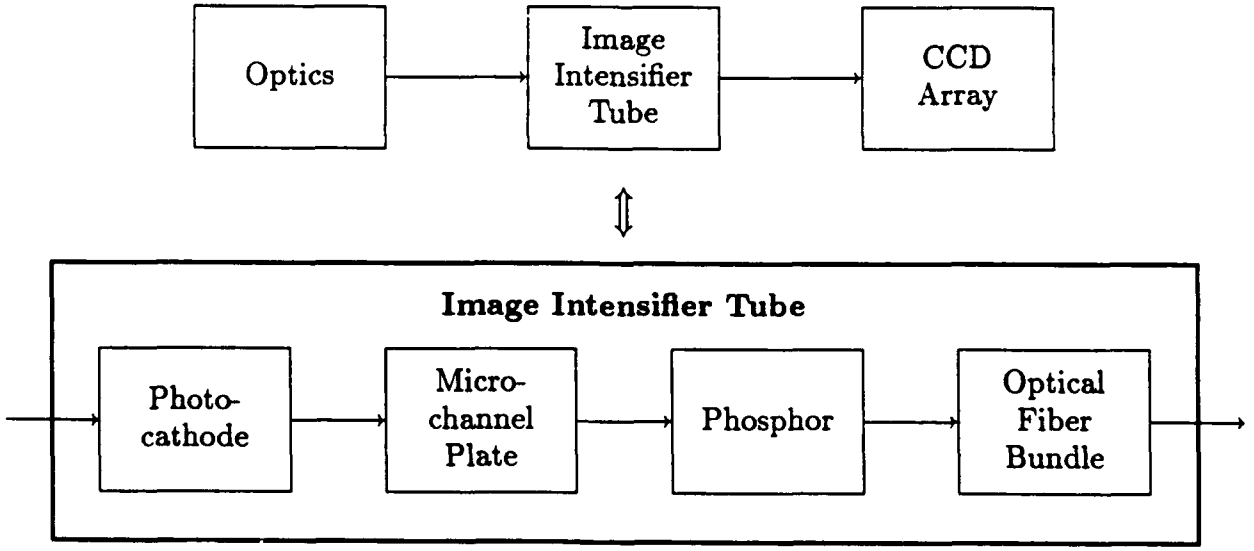


Figure 2. A typical detection system.

codes, as discussed below. Under the current project, the authors implemented a prototype of the new detection system model for angle-angle imaging systems. This prototype, which demonstrates the viability of the model, provides a test platform for further refinement and enhancement. The prototype code, written in FORTRAN '77 with minor extensions for compatibility with the SENSORSIM and DELTAS codes, runs on IBM and compatible personal computers.

The current prototype code is driven by text files, allowing the user to create component libraries with any standard text file editor. In a production version, a graphical user interface can insulate the user from the details of the text-based interface. This graphical user interface will provide tools for defining operations, represented as labeled blocks, and for defining the flow of signals through the simulation using arrows to connect the blocks. Additional mathematical routines added to the framework created during the current program will permit simulation of a larger variety of detection and processing systems. Ultimately, this interface will provide the ability to create block diagrams of any system directly on the screen. A program script, created by the graphical user interface, will provide the ability to simulate each system. This general, powerful tool will be useful for applications far beyond simulation of detection systems.

2. Considerations Affecting the Design of the Model

Several considerations affect the design of any model of optical detection systems. These considerations include the design and mission of the sensor system(s) that it must simulate, the physical characteristics of various components in detection systems, the purpose of the simulation, and software issues affecting the implementation of the model and its integration into a simulation of the sensor system.

2.1. Design of Sensor System

The SENSORSIM and DELTAS codes simulate various sensor systems from angle-angle and range-Doppler imagers to quad cell trackers and range signature sensors with single detector elements, and SENSORSIM also simulates signatures from passive (solar and thermal) sensors. These sensors sample the signal from the target at various combinations of range (time) and crossrange resolution appropriate to the type of signature that they record. Sensors that obtain angle-angle images may have either scanning (linear) or staring (planar) arrays, and some recent designs scan a single detector element over a two dimensional field of regard [4]. Effects that are negligible in one design may be significant in another, so a truly comprehensive model must correctly simulate each effect of the detection system.

2.2. Component Characteristics

There are very few components of a real detection system that do not degrade an image or a signature to some degree as compared to an ideal detection system, and many components have multiple sources of degradation. The detector model must correctly simulate all sources of degradation that affect the detected image or signature including attenuation, noise from various sources, gain, crosstalk, saturation, blooming, response time, and even the physical dimensions and materials of each component.

2.3. Goal of Simulation

The goal of a simulation governs the selection of phenomenological models to achieve the desired results, particularly in the treatment of random phenomena. The SENSORSIM and DELTAS codes both simulate instances from the same signature distribution obtained from an actual sensor system. These simulated signatures are suitable for development and evaluation of signal and image processing systems [5]. The model must draw random values from the actual statistical distribution corresponding to each noise source to achieve this goal because a limited set of statistical parameters, such as mean (expected value) and standard deviation, do not reliably characterize the composite distribution of multiple noise sources in a detection system.

2.4. User Community

The SENSORSIM and DELTAS codes serve a diverse user community. At one extreme, component designers need to modify the details of component specifications to assess the effects of minor design changes on component and system performance. At the other extreme, system analysts may want to compare the performance of representative detection systems of several different types. In the middle, system designers may want to evaluate the performance of several different components in a particular detection system configuration. The design of the new detection system model supports allows it to support all of these users.

2.5. Software Implementation and Integration

The DELTAS and SENSORSIM codes impose two constraints on the design and implementation of the detection system model. First, the model must run on a variety of host platforms (DELTAS) including an IBM or compatible personal computer (SENSORSIM). Second, the new model must work with existing models of other components such as the receiver optics and the signal processor. These considerations require the use of programming languages and data structures that are compatible with the SENSORSIM and DELTAS simulations.

3. Concept of Detection System Model

A typical detection system contains a sequence of optical, electronic, and hybrid devices that perform various mathematical operations, some desirable and others undesirable, on the incoming signal. This structure provides a sound basis for a hierarchical model in which the top level is the overall detection system, the second level represents the individual devices, and so on to the lowest level which contains individual mathematical operations performed by the various devices or elements.

The devices contained in detection systems fall into two major classes. Compound devices, like the image intensifier tube in Figure 2, contain several distinct elements that operate sequentially on the incoming signal. Simple devices, like the CCD array, do not have distinct elements. The smallest distinct physical entities in a device break down into a sequence of mathematical operations. Figure 3 shows a decomposition into mathematical operations for the photocathode of the image intensifier tube in Figure 2.

The detection system model provides separate general hierarchical representations for simple and compound devices, as illustrated in Figure 4. This hierarchical representation allows many detection systems to use a single definition of a device. Similarly, several compound devices may share a single definition of an element. This model allows users to assemble detection systems rapidly from libraries of device definitions.

The prototype code supplies the mathematical operations of addition of bias, linear and non-linear amplification, attenuation by Bernoulli selection (processes such as quantum, collection, and transmission losses), discrete pixel sampling, blurring (uniform and non-uniform), and conversion between photons and electrons. Each operation draws a random value for each pixel from the statistical distribution that characterizes the underlying physical processes, as described in Appendix A, simulating each possible response of the device or element with the correct probability. This process ensures that ensembles of images produced by the simulation have the same statistical distribution in each pixel as equivalent ensembles of images recorded by an actual sensor.

The mathematical operations at the lowest levels of the hierarchy are built into the simulation code. Standard text files, following the intuitive syntax described in Appendix B, define all other levels of the model's hierarchy. Users can edit all input files with familiar text file editors to create or modify all definitions. Users can also print text files directly and transfer them electronically using standard system resources. The simulation code interprets these files as it simulates the detection system.

The detection system model has a sophisticated numerical expression evaluator that accepts standard algebraic expressions for all floating point values in its input files. The algebraic expressions may reference variables defined in the input files as well as several system parameters, fundamental constants, and conversion factors defined by the simulation. The evaluator also provides a many standard mathematical functions, pseudorandom numbers with both uniform and Gaussian distributions, and a function that interpolates tabulated data from laboratory measurements or other sources.

The numerical expression evaluator, variables defined by the user, and the values defined by the simulation provide several important capabilites for the detection system model. The simulation defines a factor to convert responsivity to quantum efficiency, allowing the attenuation operation to use either value to simulate the quantum loss of a

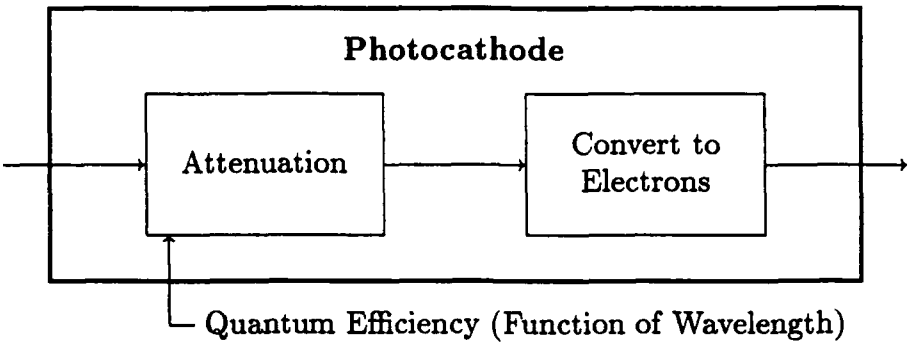
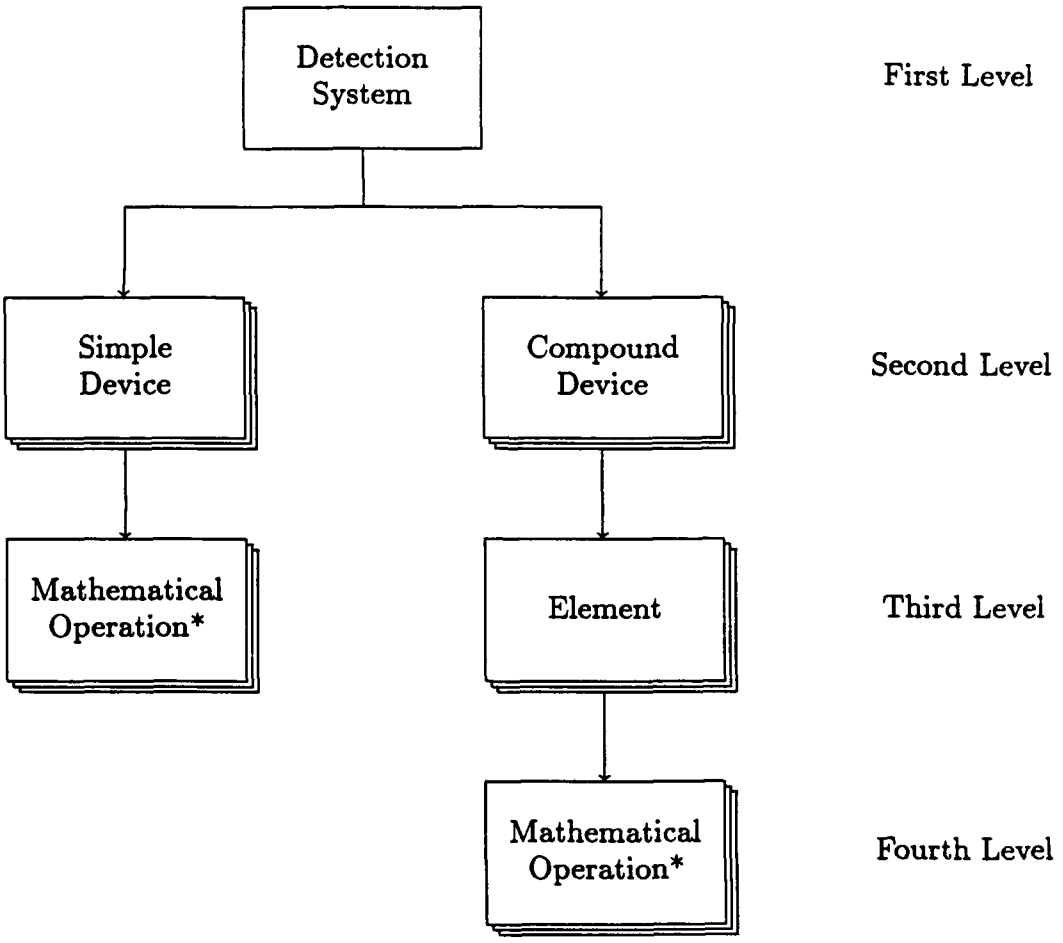


Figure 3. Mathematical definition of a photocathode.



* — Mathematical operations are provided by the simulation.

Figure 4. Hierarchical model of detection systems.

photocathode. The simulation can also interpolate either value as a function of wavelength from tabulated measurements. Similarly, the a system or compound device definition can set variables referenced in device or element defintions to the values of parameters such as supply or bias voltages that are controlled from outside the device or element. This feature allows systems with different values of such parameters to share a single definition of the device or element. Argument declarations in device and element definitions ensure that all such parameters are properly defined.

The prototype code for the detection system model, described in Appendix C, also conforms to its hierarchial structure. The code has separate subroutines for each level of the hierarchy and for each mathematical operation. Several utility routines and modules manage the image buffer, parse user input, generate random numbers with various distributions, maintain the current value of user variables, facilitate file access, and perform other incidental tasks for the simulation. This modular design ensures that the code will be easy to maintain.

The design of the detection system model and its prototype code allows for future expansion by addition of both new classes of devices and new mathematical operations. Due to time constraints on the development cycle, the prototype code does not provide mathematical operations for effects such as pattern noise that vary from pixel to pixel, but a production release should provide these operations. New devices may also perform mathematical operations that now remain unforeseen. Additional device classes might include a class for preassembled subsystems containing several devices and/or classes for specific types of devices to permit optimization of certain sequences of mathematical operations, though the need for such device classes is not now apparent. The model's expandability in these areas guarantees that it will not become obsolete as new devices appear.

4. Output from the Detection System Model

The detection system model simulates each mathematical operation that occurs in the detection system to produce a realistic instance of an image from a sensor system. Figure 5 shows the image of a cube illuminated by an incoherent laser after each step in the simulation process. In this example, the detection system consists of a simple CCD array. The detection system model executes three mathematical operations, after quantizing the image, to simulate this device. The first operation maps the image onto the pixels of the CCD array. The second operation attenuates the pixel values by Bernoulli selection to simulate quantum losses with a quantum efficiency of sixty percent. The third operation adds bias with a Poisson distribution having a mean of ten counts to simulate the CCD's read-out noise. The gray scales of these images are approximately proportional to the expected signal level to facilitate visual comparison of the images, so the mathematical operations performed by the simulation are the primary source of any differences between successive images.

Figure 6 shows images of the same scene as Figure 5 with two different levels of illumination and two different pixel configurations. Brighter illumination by a factor of ten reduces shot noise considerably in the right images compared to the left images. Larger pixels in the top images also reduce the level of noise compared to the bottom images by spatially averaging the incident signal. The gray scale of each image is proportional to the expected signal levels, but each image has the same level of read-out bias. This bias is more significant in images with weaker signal levels, causing a difference in brightness that is most pronounced in the lower left image of the figure.

Figure 7 shows a reentry vehicle observed by sensors with identical illumination and receiver optics but different detection systems. The sensor on the left has a low noise CCD detector with a conventional shutter. The sensor on the right uses a Pockels Cell as a shuttering device. The attenuation in the Pockels Cell reduces the light levels at the sensor, resulting in a higher signal to noise ratio. With consistent gray scales, the additional degradation appears as higher noise levels in the printed images.

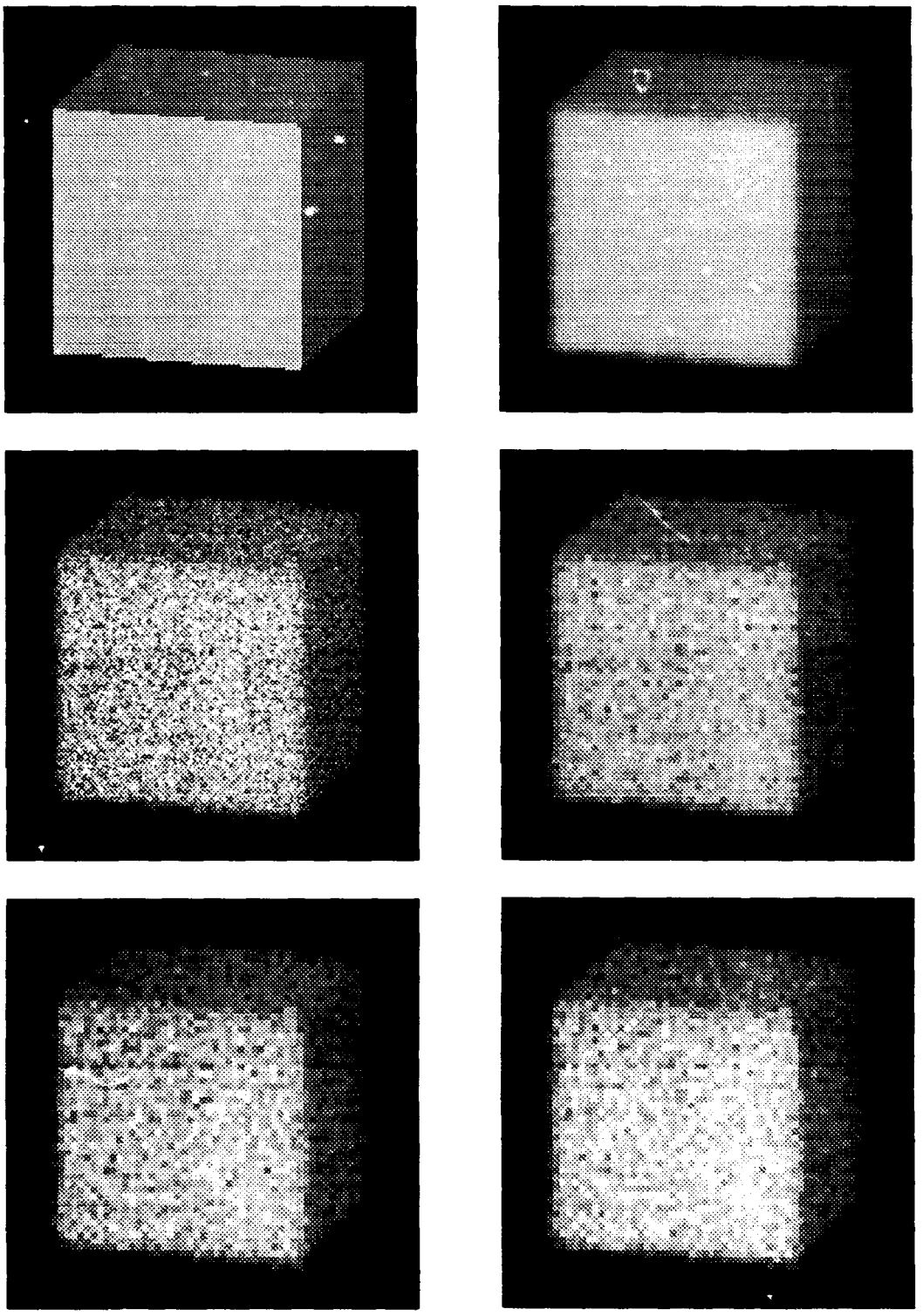


Figure 5. Simulation of an image detected by a CCD array: ideal image (top left), focal plane image (top right), shot noise (middle left), pixel geometry (middle right), quantum losses (bottom left), and read-out noise (bottom right).

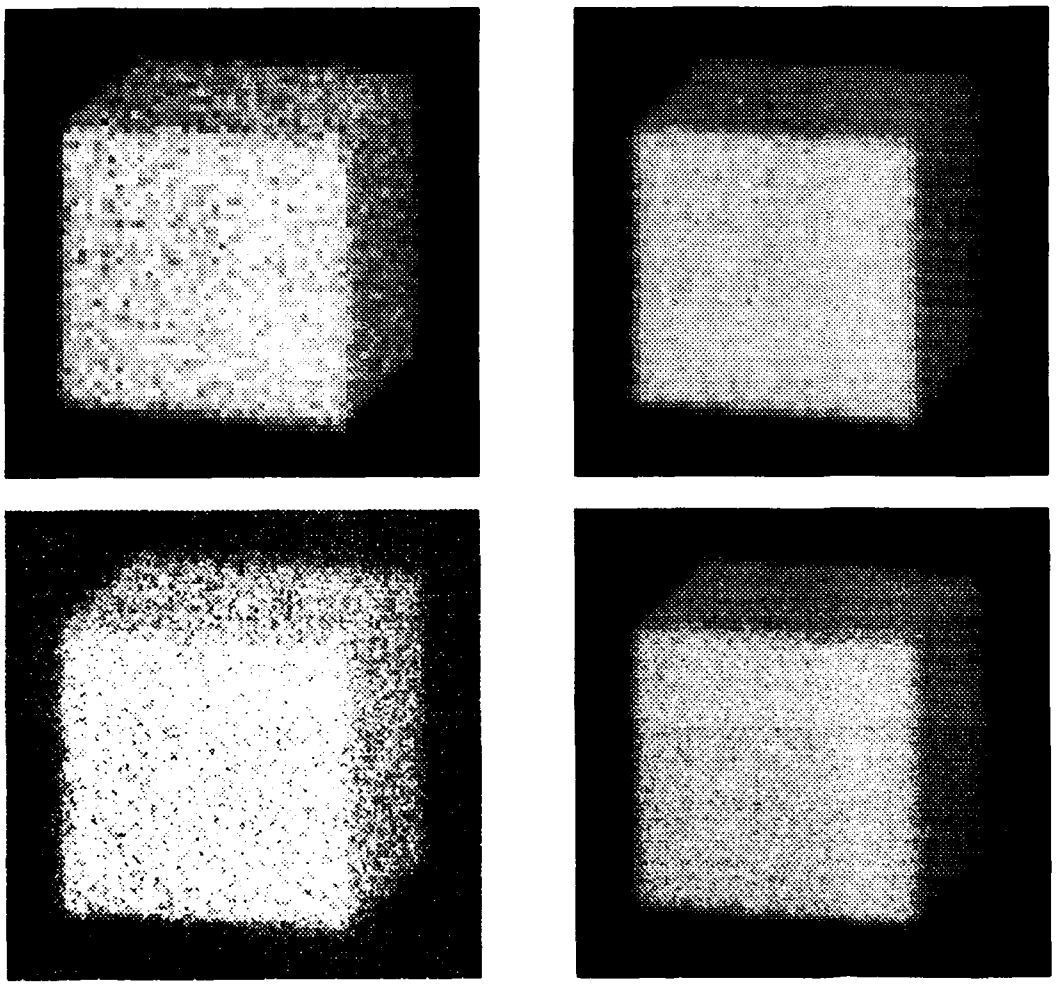


Figure 6. Effect of signal strength and pixel size on detected images. The right images have ten times as much light as the respective left images, with gray scales adjusted by a factor of ten for consistent visual contrast. The bottom images have four times as many pixels as the respective top images gray scales adjusted by a factor of four, again to maintain consistent visual contrast.

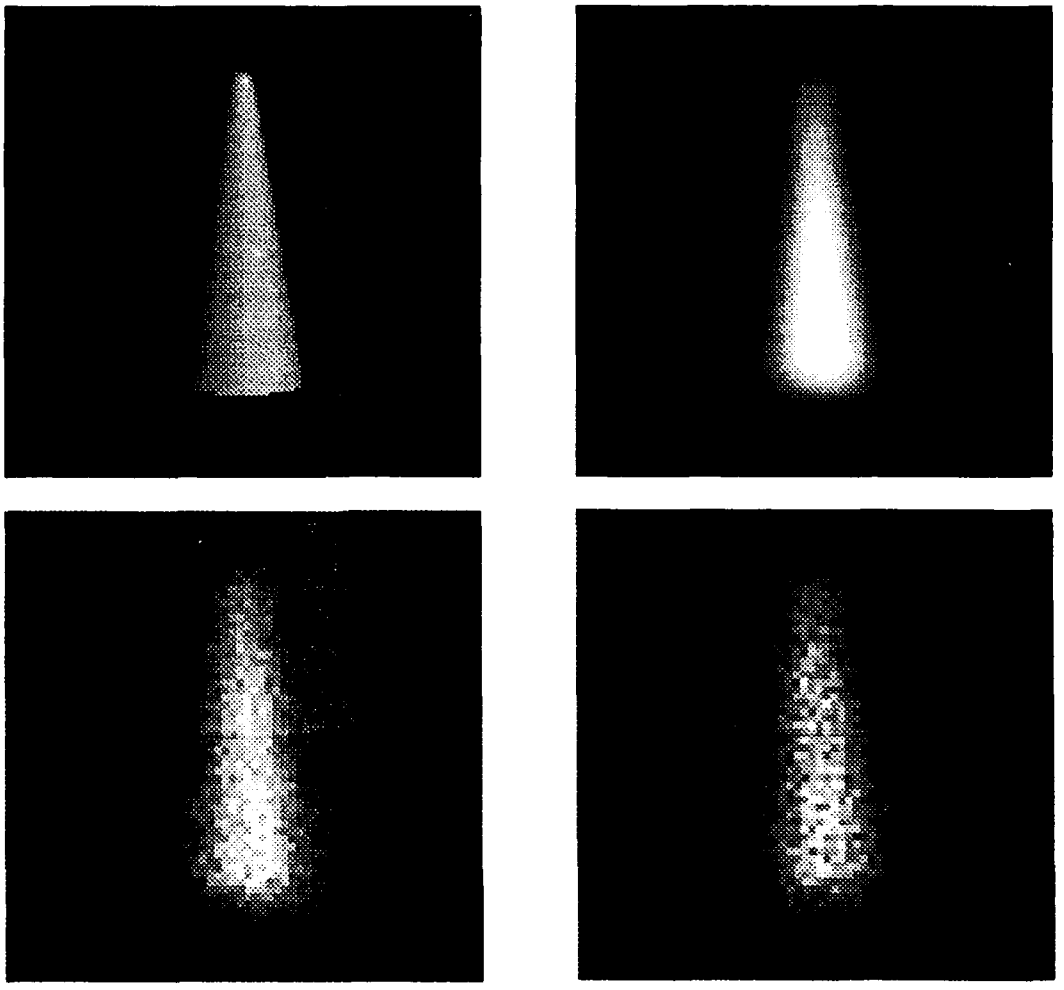


Figure 7. SENSORSIM Images of a generic reentry vehicle illuminated by an incoherent laser: ideal image (top left), focal plane image (top right), and detected images from two detection systems (bottom).

5. Further Development of Detection System Model

The successful implementation of a prototype code has demonstrated the viability of SPARTA's comprehensive model of optical detection systems. The prototype code is also available for use as a test platform for further enhancement and refinement of the model.

Time constraints on the development of the prototype code precluded implementation of mathematical operations to simulate pattern noise, which may be either multiplicative (non-uniform gain or attenuation) or additive (non-uniform bias). These phenomena frequently affect imaging sensors. Further work on the detection system model should include addition of these operations. The prototype code already contains the random number generators and the image buffer manager functions to support these operations, so the time required to add these operations will be minimal.

The prototype code does not simulate sensor systems that sample temporally rather than spatially. Range-Doppler imagers and quad cell (3-D) tracking systems are among the standard system designs that record temporal samples. With some modification to the data structures in the prototype code, the detection system model can also simulate these systems. The authors recommend addition of this capability to the production version of the detection system model.

A preprocessor to create and edit files defining detection systems, devices, and elements would greatly enhance the detection system model. This preprocessor might use a graphical interface to show each definition as a block diagram. The user would select a device or an element with a mouse or other pointing device to view or edit its definition.

With these enhancements, the detection system model should be integrated into the DELTAS code. This integration will entail minor modification of some routines to use the DELTAS data structures. The project should also include development of libraries of files containing definitions of a selection of devices representing the state of the art at the principal wavelengths of current laser radar systems.

The development of the detection system model also has implications for other aspects of optical sensor simulations, as described below. These possibilities should also be pursued as availiable funding allows.

6. Other Implications of Technological Developments

The technology developed for the detection system model has several implications for the simulation of optical sensor systems. The most immediate benefit is, of course, a truly useful and comprehensive model of optical detection systems, but several technologies developed to support the detection system model are critical to advances in other areas as well.

6.1. Hierarchical Models

The concept of a hierarchical model controlled by a flexible command structure is a major advancement in the design of phenomenological models for optical sensor systems. This concept has its roots in SPARTA's Basic Scene Descripton Language (BSDL) and Basic Target Description Language (BTDL) of the SENSORSIM and DELTAS codes. Both of these languages provide a nestable `include` command to execute a sequence of commands in another file, but the detection system model is the first to define a rigorous structure for the hierarchy.

SPARTA can also develop an advanced hierarchical model of the sensor's signal and image processing system. This application requires a set of basic mathematical operations that simulate the response characteristics, including noise, of analog and digital electronic elements. The current DELTAS simulation contains a preliminary signal processing model based on this concept that has proven the viability of this concept, but the final DELTAS Signal Processing Model, scheduled for completion during the option years of the DELTAS project, never received funding. Advances implemented in the prototype code for the detection system model, such as the numerical expression evaluator, the user variable handler, and the image buffer manager, provide the base of technology to expand signal and image processing capability beyond that envisioned for the final DELTAS model. The authors believe that this enhanced signal and image processing capability would be a useful addition to the DELTAS code.

6.2. Numerical Expression Evaluator

The numerical expression evaluator in the prototype code opens the door to several major enhancements to the SENSORSIM and DELTAS codes. Perhaps the most obvious of these enhancments is an extension of SPARTA's BSDL and BTDL to allow expressions for numerical arguments. This enhancement would allow the scenes and targets to change as a function of certain values defined by the simulation or of values defined in a scene parameter file supplied by the user. The scene compiler also provides random number functions with uniform and Gaussian distributions that might facilitate simulation of irregular objects.

The numerical expression evaluator can also evaluate mapped attributes in scene or target descriptions. The present versions of the SENSORISM and DELTAS simulations do not support mapped attributes. Either simulation could store the definitions of mapping functions in character strings during scene compilation, then pass these definitions to the numerical expression evaluator to obtain the texture at visible points on the object surface after defining variables such as the coordinates of the point on the surface. Mapped attributes can include vibration and texture, the former affecting the Doppler signature of a target and the latter simulating camouflage paint schemes.

A logical expression evaluator, following the same design as the numerical expression evaluator, would allow programming constructs for conditional execution of blocks of commands. Added to BSDL or BTDL, for example, a file defining an inflatable decoy might define the shape of the inflated decoy after its inflation time or the shape of its canister before its inflation time.

7. Summary and Conclusions

The authors have developed a comprehensive model of optical detection systems that is compatible with both SPARTA's optical sensor simulation, SENSORSIM, and the Defense Laser/Target Signatures (DELTAS) code. This model, which overcomes the limitations that can be quite severe in many other models, simulates physically and mathematically correct examples of images obtained from detection systems of any complexity. Under the current project, the authors implemented a working prototype of the model that demonstrates the viability of its approach. This prototype code, written in FORTRAN '77 with minor extensions, will serve as a test platform for further refinement of the model and its eventual integration into the DELTAS and SENSORSIM codes.

The detection system model follows an intuitive hierarchical structure that conforms to the design of the detection system. This structure allows system analysts to assemble detection systems very quickly from libraries of predefined components and rapidly evaluate competing designs. System designers can change components in a detection system with equal ease to evaluate the relative performance of competing devices. Component designers can modify component definitions to evaluate the effect of small design changes on device performance. The prototype code can also save intermediate images, permitting analysis of the performance of each component. Thus, this model will satisfy the requirements of a diverse user community.

Technology developed for the prototype detection system model will allow other enhancements to the DELTAS and SENSORSIM codes. The hierarchical design of the detection system model can also simulate signal and image processing systems with a similar degree of fidelity. The DELTAS simulation contains a preliminary model of this type that has considerable room for enhancement. The numerical expression evaluator and the user variable handler can provide similar capabilities for scene and target descriptions and a mechanism for mapping textures and other attributes onto target surfaces. A logical expression evaluator and a logical flag handler based on the same algorithms would allow conditional execution of commands through "block `if`" command structures. These capabilities will significantly enhance the SENSORSIM and DELTAS codes.

8. References

1. N. R. Guivens, Jr. and P. D. Henshaw, "Image Processor Development with Synthetic Images," in D. J. Svetkoff, Ed., *Optics, Illumination, and Image Sensing for Machine Vision IV*, Proc. SPIE **1416** (1991), pp. 167–180.
2. N. R. Guivens, Jr., P. D. Henshaw, R. Marino, and K. Schultz, "Detection Systems for a Common Module Tracker (U)," Proc. IRIS Specialty Group on Active Systems, 16–18 October 1990 (SECRET).
3. N. R. Guivens, Jr. and P. D. Henshaw, "A Comprehensive Model of Optical Detection Systems," in R. J. Becherer, Ed., *Laser Radar VII: Advanced Technology for Applications*, Proc. SPIE **1633** (1992), pp. 244–255.
4. R. F. Dillon, D. P. DeGloria, F. M. Pagliughi, J. T. Muller, M. G. Cheifetz, D. E. B. Lees, and M. Michalik, "Laser Radar Imaging Experiments," in R. J. Becherer, Ed., *Laser Radar VII: Advanced Technology for Applications*, Proc. SPIE **1633** (1992), pp. 274–280.
5. N. R. Guivens, Jr. and P. D. Henshaw, "Image Processor Development with Synthetic Images," in D. J. Svetkoff, Ed., *Optics, Illumination, and Image Sensing for Machine Vision IV*, Proc. SPIE **1416** (1991), pp. 167–180.

Appendix A. Noise in the Detection System

Images and signals from optical sensors contain noise from numerous sources characterized by various statistical distributions. The detection system model draws a random value for each pixel from the correct statistical distribution for each source of noise in the detection system, beginning with shot noise on the incoming optical signal. This process cascades the statistical distributions of the noise sources in the same manner as an actual detection system, thus guaranteeing that an ensemble of simulated images exhibits the same statistical distribution in each pixel as a corresponding ensemble of images recorded by an actual sensor. This noise model ensures that processing systems developed and tested from simulated images will work reliably in operational sensors.

A.1. Basic Noise Model

The detection system model must simulate noise from various sources. The signal collected by the sensor's optics has shot noise due to random arrival of photons at the detector, and it may also have noise due to speckle if the illumination is at least partially coherent. Most elements in a detection system also add some form of noise to the signal. The binomial distribution family, illustrated in Figure A–1, characterizes these sources of noise.

The SENSORSIM and DELTAS models of the receiver optics generally determine the expected intensity of illumination in each pixel of the detection plane *for a single speckle pattern.* Several executions of the detection system model with a single signature from the optics model, therefore, represent multiple measurements of a static speckle pattern, as in a laboratory experiment in which rigid mountings prevent motion of the sensor and the target(s). In this case, only shot noise and noise within the detection system cause differences in the detected images. Of course, with incoherent illumination, the optics models generate the expected intensity without speckle.

Most generally, the receiver optics model generates values from a gamma distribution in each pixel [1] for the case of partial coherence.* The gamma distribution has the form

$$p_{n'}(x) = \left(\frac{n'}{\mu}\right)^{n'} \frac{x^{n'-1} e^{-n'x/\mu}}{\Gamma(n')} \tag{1}$$

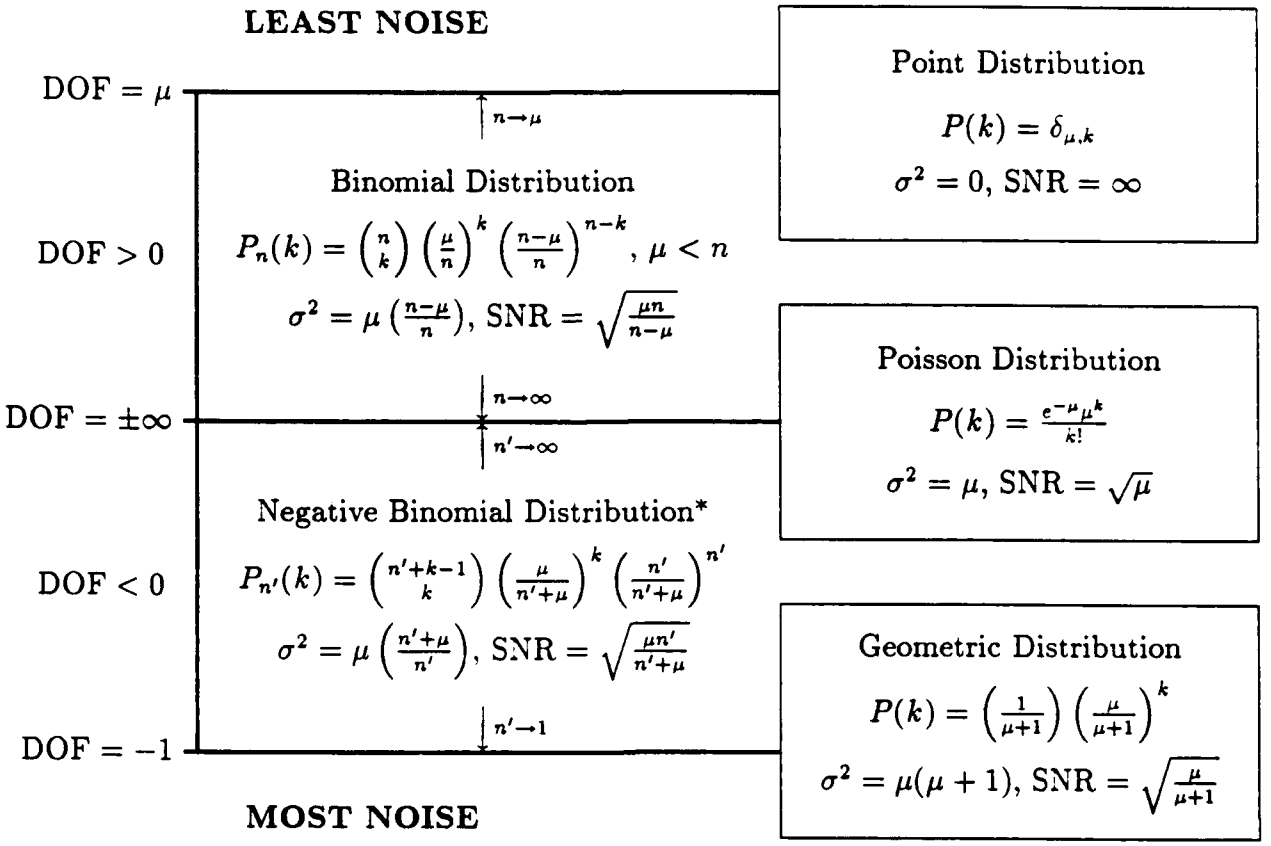
where

x is the expected signal for the speckle pattern,

n' is the freedom parameter (number of "coherence cells"), and

μ is the *a priori* expected signal, without knowledge of the speckle pattern.

* Although the laser beam and receiver optics models in the SENSORSIM and DELTAS codes treat only the limiting cases of coherent or incoherent illumination, the case of partial coherence may arise at the detector plane through incoherent integration of a coherent signal over the active area of each pixel in either simulation. Both simulations handle this situation correctly without explicitly computing the freedom parameter of the gamma distribution.



* — Binomial Distribution with $n = -n'$

Figure A-1. The binomial family of probability distributions. The expected (mean) value for each distribution is μ and the freedom parameter is n or $-n'$.

The gamma distribution, with a variance of μ^2/n', reduces to the negative exponential distribution

$$p_1(x) = \frac{1}{\mu}e^{-x/\mu} \tag{2}$$

in the coherent limit ($n' = 1$). In the incoherent limit ($n' \to \infty$), the gamma distribution converges to

$$p_\infty(x) = \delta(x - \mu) \tag{3}$$

where $\delta(x)$ is Dirac's delta function.

The detection system model draws a random value from a Poisson distribution with a mean equal to the expected signal, in counts, to simulate the shot noise in each pixel at the detection plane. This quantization of the signal combines the gamma distribution of the expected incident signal with a Poisson distribution by the cascade relationship

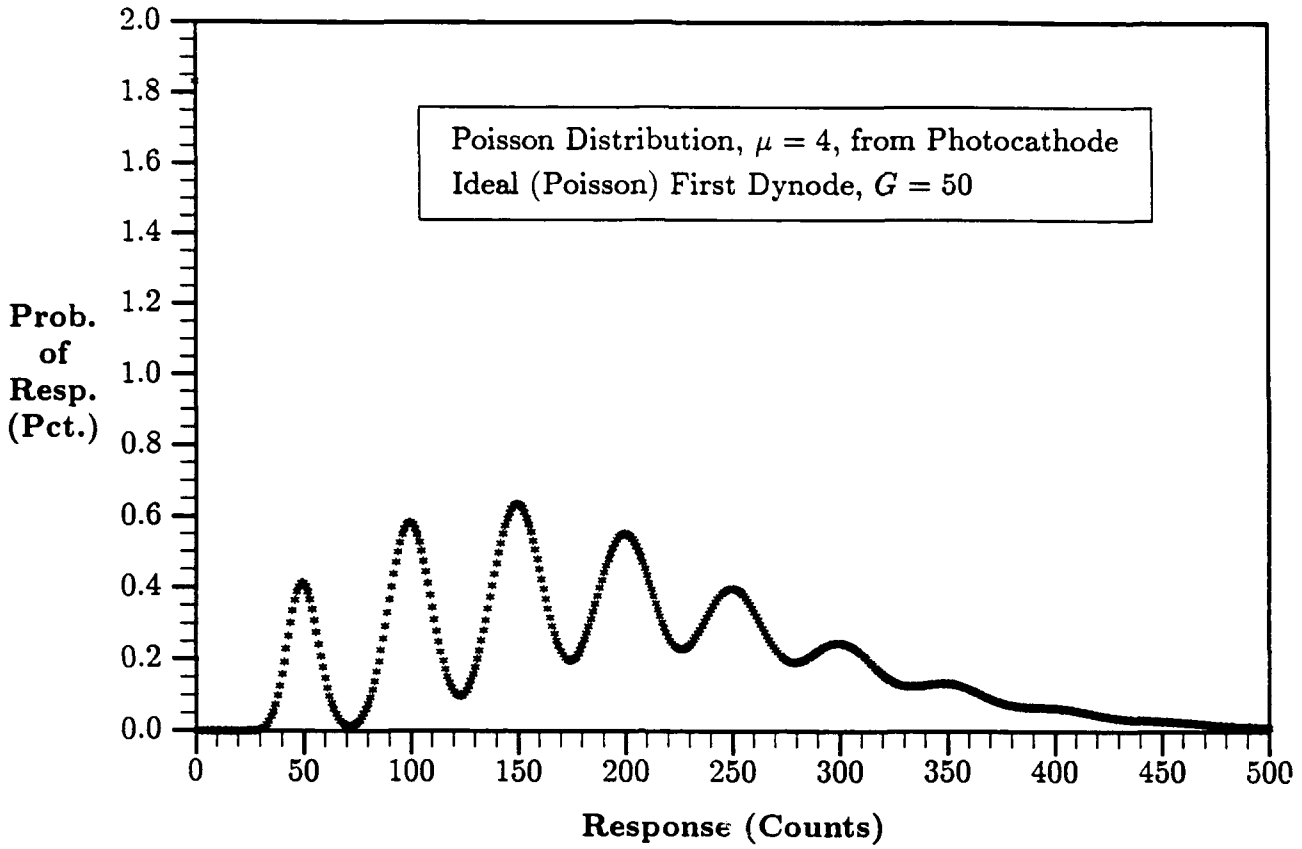


Figure A–2. Statistical distribution for cascade of two Poisson elements. The first element has an expected value of 4. *The second element has a gain of* 50.

$$P_{n'}(k) = \int_0^\infty p_{n'}(x) \left(\frac{e^{-x} x^k}{k!} \right) dx \tag{4}$$

where the expected value of the Poisson distribution is the value drawn from the gamma distribution. Upon substitution for $p_{n'}(x)$ by equation (1) and evaluation of the integral, equation (4) reduces to the formulation of the negative binomial distribution in Figure A–1. Equation (4) similarly reduces to the exponential and Poisson distributions for the respective limiting cases in equations (2) and (3).

Once the detection system model quantizes the signal to simulate shot noise, it maintains all signal values in counts of photons or electrons. This choice of units allows the detection system model to simulate each source of noise by a drawing pseudorandom value for each pixel from the correct distribution for each source of noise in Figure A–1. This process guarantees that the values in each pixel have the same statistical distribution as images obtained from an actual sensor. Figure A–2, which is very similar to published response curves for photomultiplier tubes [2], shows the effect of cascading two Poisson noise sources representing the photocathode and first dynode of a photomultiplier tube with very low noise. Standard probability functions cannot approximate the resulting multimodal distribution which this model reproduces correctly.

Multiplicative operations in a detection system include amplification and attenuation. In these operations, the expected output signal (μ) is always the product of a gain or attenuation (efficiency) factor and the input signal. The output signal is always the value drawn from the correct distribution in Figure A-1.

Many elements in a detection system attenuate the signal by Bernoulli selection, characterized by the binomial distribution. In a Bernolli process, each count (photon or electron) entering a stage has a certain probability of exiting independent of what happens to other counts. Since each count of the input signal represents an independent event, the number of degrees of freedom (n) is always the number of counts in the input signal. Examples of Bernoulli processes include transmission, collection, and quantum losses.

The negative binomial distribution generally characterizes amplifying operations in a detection system. The freedom parameter of the distribution (n') controls the severity of noise in the detection system. The freedom parameter is always proportional to the number of counts in the input signal, so the user need only specify the value for a single count of input. The simulation automatically multiplies this value, as well as the gain, by the number of counts of input. Microchannel plate image intensifiers and dynodes in a photomultiplier tube are examples of amplifying devices.

Sources of bias, like sources of amplification, are generally characterized by the negative binomial distribution, or by the Poisson distribution in the limit of low noise. The bias value drawn from this distribution, however, is added to the signal already present in the pixel. Since the level of bias is independent of the input signal in the pixel, the simulation does not scale the distribution parameters. Sources of bias include dark currents and CCD readout noise.

A.2. Effects of Pixel Geometry and Coupling

Standard detection system components have various geometric structures. Some devices or elements have distinct pixels; others do not. The prototype code supports circular, square, rectangular, and hexagonal pixels on square, rectangular, and hexagonal grids. These options represent the vast majority of standard components, but the authors designed the prototype code to allow addition of other configurations if a need arises.

Devices in a detection system may change the geometry of the image plane either by tapering pixels to a new geometric arrangement or by resampling the signal. The tapering operation retains the signal level in each element as it changes the size and/or location of the individual elements, whereas a resampling operation redistributes the signal in each pixel among overlapping pixels and gaps of the new geometric arrangement. Devices and elements with clearly defined geometric structures normally resample the incident image. Some devices or elements, such as a tapered fiber optic coupler, may both resample and taper the image. In the detection system shown in Figure 2, the microchannel plate, the fiber optic bundle, and the CCD array all resample the image. The photocathode and the phosphor do not resample the image because they are continuous materials with no clearly defined pixel structure.

The prototype model adjusts internal parameters describing the pixel geometry to simulate taper devices, but it does not change the values of the any pixels or the dimensions of the array. The model can apply any attenuation, gain, bias, or crosstalk due to the taper

mechanism separately either before or after the taper since the taper operation does not affect the pixel values.

The resampling operation is more complicated than the tapering operation because the detection system model must actually resample the image or signal. The model allocates a new buffer for the resampled image, then it distributes the counts in each pixel of the old buffer to the overlapping pixels of the new buffer by a series of binomial distributions. For each overlapping pixel, the parameters of the binomial distribution are

$$\mu = \frac{n_r A_\cap}{A_r} \tag{5a}$$

$$n = n_r \tag{5b}$$

where

n_r is the number of counts that remain unassigned,

$A_\cap$ is the area of the intersection of the old pixel and the current new pixel, and

A_r is the area of the old pixel not intersected by previous new pixels.

This process, illustrated in Figure A–3, follows immediately from the assumption that the remaining signal counts of the original pixel are dispersed with a uniform distribution in the area that remains unintersected by new pixels. Counts of the old pixel that remain after sampling all intersecting new pixels are lost in the gaps between the pixels of the new geometry. This resampling operation can introduce considerable noise into an image.

A similar process of redistributing the signal in a pixel simulates phenomena like crosstalk that partially couple nearby pixels. A convolution kernel specifies the probabilities associated with each possible offset from the original pixel. The simulation draws the actual counts redistributed to each offset from a binomial distribution in the same manner as in the resampling operation. For the i^{th} entry in the kernel, the distribution parameters are

$$\mu = \frac{n_r p_i}{1 - \sum_{j=1}^{i-1} p_j} \tag{6a}$$

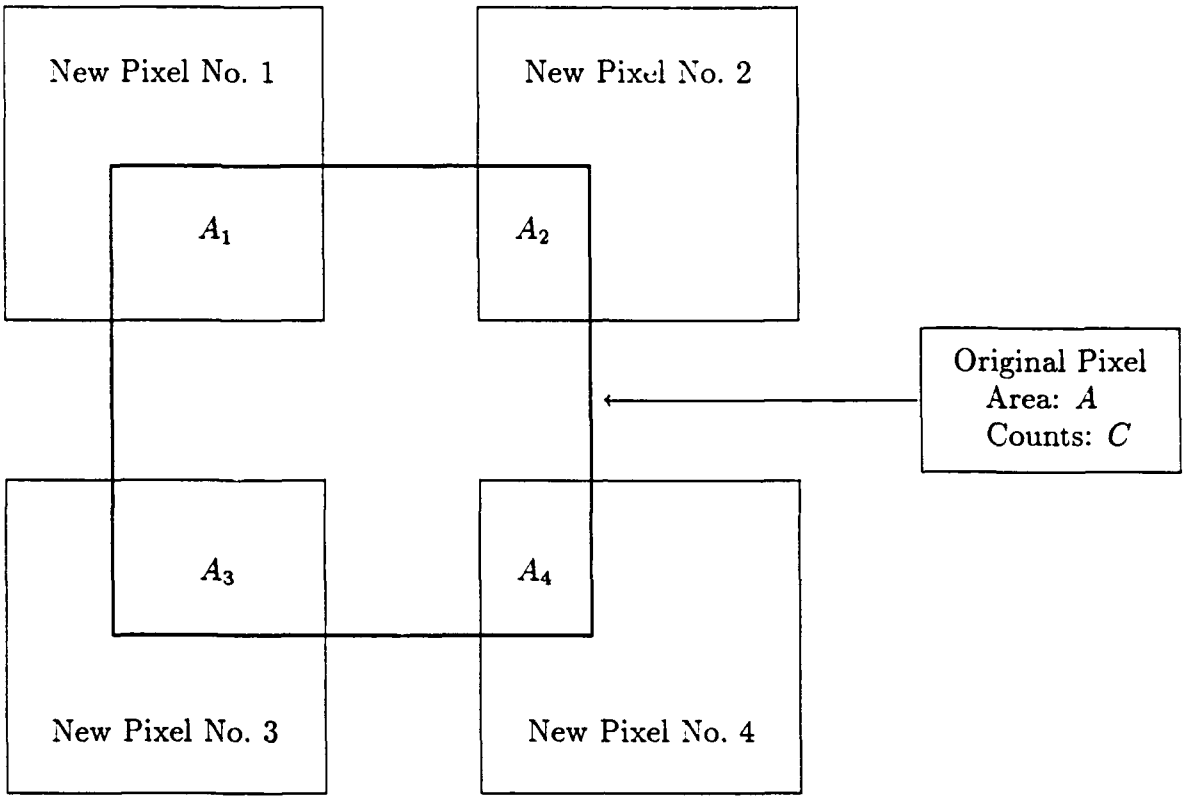
$$n = n_r \tag{6b}$$

where

p_i is the probability that a count is shifted to the new pixel and

n_r is the number of counts remaining in the original pixel.

The simulation retains counts any counts that remain after completing this dispersion process in their original pixel, so signal loss occurs only when the shift causes the new pixel to fall beyond the bounds of the image array. This process can also introduce noise into the image.



Step 1. Random Draw for Counts in New Pixel No. 1, C_1 Binomial Distribution, $n = C$, $\mu = \frac{nA_1}{A}$
Step 2. Random Draw for Counts in New Pixel No. 2, C_2 Binomial Distribution $n = C - C_1$, $\mu = \frac{nA_2}{A-A_1}$
Step 3. Random Draw for Counts in New Pixel No. 3, C_3 Binomial Distribution, $n = C - C_1 - C_2$, $\mu = \frac{nA_3}{A-A_1-A_2}$
Step 4. Random Draw for Counts in New Pixel No. 4, C_4 Binomial Distribution, $n = C - C_1 - C_2 - C_3$, $\mu = \frac{nA_4}{A-A_1-A_2-A_3}$

Figure A–9. Resampling a pixel.

A.3. References

1. J. W. Goodman, *Statistical Optics*, Wiley & Sons, New York, 1985.
2. See, for example, R. W. Engstrom, *Photomultiplier Handbook*, RCA Corporation, Lancaster, PA, 1980.

Table B–I. Default Filename Extensions for Prototype Model.

Defined Entity	Default Type
Detection System	.DET
Simple Component	.SMP
Compound Component	.CMP
Element of Compound Component	.ELT

Appendix B. Definition of Detection Systems and Components

The definition of a detection system follows the hierarchical structure of the detection system model, omitting the lowest level. Each block above the lowest level on a block diagram represents a separate input file, with one additional file to specify the sequence of blocks at the top level. The mathematical operations at the lowest level of the model's hierarchy do not require separate input files because the simulation code contains their definitions.

The prototype code uses four types of files to define detection systems (at the top level of the hierarchy), simple components, compound components, and elements of compound components. A file defining a detection system specifies simple and compound components by the names of the files containing their definitions. A file defining a compound component likewise specifies its elements by the names of the files containing their definitions. Files defining simple components and elements of compound components specify a sequence of operations that simulates the corresponding component or element. Thus, the complete definition of a detection system consists of exactly one detection system file, a file defining each component in the detection system, whether simple or compound, and a file defining each element of any compound components. The prototype code supplies the default filename extensions shown in Table B–I if the user does not explicitly specify an extension as part of the file name. Each file name may also include a directory path if it is not the operating system default.

The input to the detection system model follows a syntax similar to SPARTA's Basic Scene Description Language (BSDL) [1] and Basic Target Description Language (BTDL) [2] in the SENSORSIM and DELTAS codes. All definition files share the following syntactic conventions with BSDL and BTDL.

- Each record of an input file contains either a command or a comment.
- Each command record contains an intuitive key word, usually followed by a specific sequence of arguments. The arguments may include key words and numerical values.
- All key words are lower case.
- Comment records, universally marked with an asterisk (*) as the first non-blank character, have no effect.
- Blank records are forbidden. (Records that contain an asterisk as the only non-blank character are valid comment records.)

Table B–II. Syntax for Specification of Detection System.

Command	Syntax
Device	`device` <*class*> <*filename*>
Set Variable	`set` <*variable*> <*value*>
Delete Variable	`delete` <*variable*>
Save Image	`save` <*filename*>
Comment	* <*comment text*>

These conventions allow the detection system model to share parsing routines and a consistent interface with the scene and target models of the SENSORSIM and DELTAS codes.

The prototype code employs variables defined in its input files to pass parameters between definitions. All variables are global entities that exist for the duration of the simulation unless explicitly deleted. The prototype model permits the user to define up to fifty (50) variables concurrently, in addition to the predefined values described below. The user may reference these variables in an algebraic expression for any argument that requires a floating point value. Each definition should delete the variables defined within it when they are no longer required to release its storage for other variables.

All syntax tables below follow standard notational practices of the computer profession. A fixed pitch (typewriter) font indicates `key words` and operators that must appear exactly as shown. Italic text in angle brackets denotes a <*description of an entity*> that the user must supply. Square brackets enclose [<*optional entities*>], and ellipses (...) indicate repeatability of the preceeding optional entity.

B.1. Specification of a Detection System

The detection system model always requires exactly one detection system file to specify the sequence of components in the detection system. Table B–II shows the command syntax for in this file.

The `device` command specifies the file containing the definition or parameters for the next device in the detection system and the class of device. In the prototype model, the argument <*class*> may be either `simple` for a simple device or `compound` for a compound device. All other values of the class identifier are reserved for additional classes of devices that may be added at a later date.

The `set` command assigns a value to a variable, creating the variable if it does not already exist. The `delete` command removes the specified variable. Any arithmetic expression in an input file may reference a variable by name while the variable exists.

The `save` command writes a copy of the current image to a disk file. The default file type is `.IMG`, indicating a SENSORSIM image file. This command allows examination of intermediate images, supporting identification of sources of degradation within the detection system and analysis of their severity.

Table B–III. Syntax for Definition of Simple Devices and Elements of Compound Devices.

Function	Command Syntax
Declare Argument	`argument` <*name*> <*description*>
Amplification	`gain` <*gain factor*> <*freedom parameter*>
Signal Loss	`atten` <*attenuation factor*>
D. C. Bias	`bias` <*expected counts*> <*freedom parameter*>
Convert to Electrons	`electron`
Convert to Photons	`photon` <*wavelength*>
Set Variable	`set` <*variable*> <*value*>
Delete Variable	`delete` <*variable*>
Comment	* <*comment text*>

Function	Command Sequence

Resample Image	`pixelmap`
	<*pixel geometry specification*>
	`endmap`

Taper Pixels	`tapermap`
	<*pixel geometry specification*>
	`endmap`

Spread Signal to Nearby Pixels	`spread`
	<*convolution kernel specification*>
	`endspread`

B.2. Definition of Simple Devices and Elements of Compound Devices

A file defining a simple device or an element of a compound device contains the sequence of mathematical operations performed by the device. Table B–III shows the syntax for the operations provided by the current prototype code.

The `argument` command, normally placed at the beginning of the definition, identifies the variable <*name*> as an argument to the definition that follows. The prototype model checks that this variable exists before processing the remainder of the definition. The <*description*>, treated as a comment by the prototype model, should briefly describe the corresponding parameter.

The `gain` command specifies an amplifying mechanism within a device or element. The value of the <*freedom parameter*> may be zero for an ideal (Poisson) amplifying element or the freedom parameter (n') of the negative binomial distribution *for a single event* (input of one count) in each element. The simulation multiplies both values by the number of events (incident counts) to obtain the actual distribution parameters.

The `atten` command specifies a loss by Bernoulli selection within the component. The

attenuation (loss) factor must be between zero and one.

The `bias` command specifies the addition of a fixed (D. C.) component to the existing signal. The value of the *<freedom parameter>* may be zero to indicate an ideal (Poisson) bias source or the freedom parameter (n') of a negative binomial distribution with the expected value specified by *<expected signal>*.

The `electron` and `photon` commands convert the signal, respectively, to electrons or to photons of the specified wavelength. The wavelength in the `photon` command must be in meters. These commands do not change the number of counts in an image pixel, but they set internal values in the model for the corresponding type of signal including the `lambda` and `resp` values described below.

The `set`, `delete`, and comment commands are identical to the corresponding commands in the detection system file. The definition of a device or element should generally "clean up after itself" by deleting any variables that it creates.

The `pixelmap...endmap` and `tapermap...endmap` command blocks specify changes in the pixel geometry. The `pixelmap` block tells the simulation to resample the image pixels by dividing the signal in an old pixel among the overlapping new pixels and interpixel spaces. The `tapermap` sequence retains all counts in the corresponding pixel while changing the pixel geometry. The key words `pixelmap` or `tapermap` and `endmap` each occupy separate records of the input file. The *<pixel geometry specification>* in these command blocks contains a separate set of commands described below.

The `spread...endspread` command block specifies a convolution kernel for redistribution of the counts in each pixel of an image. The specification of the convolution kernel is described below.

B.3. Specification of Pixel Geometry

Common components in detection systems have a number of geometric configurations in their image planes. The active area of a pixels may be square, rectangular, hexagonal, or circular. The pixels may be centered on square, rectangular, or hexagonal grids. Further, the center of a device may not be on the axis of the detection system and the device may be rotated relative to other components. The prototype model requires numerous parameters and tokens to fully specify all of these variations, exceeding the reasonable capacity of a single command line.

At first glance, a pixel geometry specification in separate file referenced by a single command of the form

`pixelmap` *<file>*

seems more consistent with the convention of one command per line to which the rest of the detection system model strictly conforms. In requiring a separate file for every change of pixel geometry, and thus for every device or element with a distinct pixel structure, this approach would be a serious inconvenience in maintaining libraries of devices and in distributing definitions to other users. The geometric characteristics of a device or element logically belong in the device definition rather than a separate file because they are inherent in the device or element itself. The pixel geometry command block embedded in the simple device or element definition is therefore a more satisfactory solution. Similar reasoning applies to the embedding of convolution kernels in `spread...endspread` command blocks.

Table B–IV. Syntax for Specification of Pixel Geometry.

Function	Command Syntax
Array Size†	`array` <*rows*> <*columns*>
Active Pixel Area	`pixel` <*shape specifier*>
Grid Geometry	`grid` <*grid specifier*>
Grid Position	`center` <*row shift*> <*column shift*>
Grid Orientation	`orient` <*orientation angle*>
Comment	* <*comment text*>

† — Not allowed in `tapermap...endmap` command blocks.

Table B–V. Pixel Shape Specifiers.

Shape	Specifier
Square	`square` <*width* >
Rectangle	`rect` <*height*> <*width*>
Hexagon	`hex` <*width*>
Circle	`circle` <*diameter*>

Table B–IV shows the pixel geometry commands that may appear in the <*pixel geometry specification*> of a `pixelmap...endmap` or `tapermap...endmap` command block. These commands may appear in any sequence. If a command is not present, its associated values do not change in the mapping operation.

The `array` command specifies the number of rows and columns in the pixel array. The respective arguments <*rows*> and <*columns*> must have integer values. For non-rectangular grids, the number of columns is the number of elements in each row. The `array` command is forbidden in `tapermap...endmap` command blocks because tapering devices cannot change the number of pixels in the image array.

The `pixel` command defines the active area of the pixel. Table B–V shows the permitted values for the <*shape specifier*>. The hexagon is oriented with two sides perpendicular to the rows of the grid, and the <*width*> is measured perpendicular to two opposite sides. The dimensions in the `pixel` command generally should not exceed the corresponding spacing parameters in the `grid` command.

The `grid` command specifies the grid of points marking the centers of the image pixels. Table B–VI shows the values permitted for the <*grid specifier*>. Spacing parameters are always measured from the center of one pixel to the center of the next, and thus are the largest value permitted for the corresponding dimension in the `pixel` command.

Hexagonal grids are somewhat more complicated, mathematically, than square and rectangular grids. The odd rows of pixels are shifted to the left by one half of the pixel separation relative to the even rows, effectively creating twice the specified number of columns but locating pixels only at "even" grid points (that is, grid points where the dif-

Table B–VI. Grid Specifiers.

Shape	Specifier
Square	`square` <*spacing*>
Rectangle	`rect` <*row spacing*> <*column spacing*>
Hexagon	`hex` <*spacing*>
Offset	`offset` <*modulus*> <*row spacing*> <*column spacing*>

ference between the row and column indices is even). The simulation adjusts the separation between the rows of pixels to maintain the same separation between adjacent pixels along the resulting diagonals as along the rows. Figure B–1 shows hexagonal and rectangular grids with the same spacing and dimensions.

The offset grid extends of the concept of the hexagonal grid by dividing the image into groups of <*modulus*> rows. The corresponding rows in each group are shifted a proportionate amount to create a diagonal alignment, so that each diagonal column of one group aligns with the adjacent diagonal column of the next group. The diagonals run from upper left to lower right if <*modulus*> is positive or from upper right to lower left if <*modulus*> is negative. The <*column spacing*> parameter for an offset grid is the distance between adjacent pixels within each row, so that <*row spacing*> and <*column spacing*> specify the respective maximum dimensions of rectangular pixels. The command

```
grid hex spacing
```

is equivalent to

```
grid offset 2 spacing/sqrt(3.0) spacing
```

where `spacing` is a variable set to the desired grid size.

The `center` command specifies the location of the center of the pixel grid in meters along the reference axes (the axes of the original image).

The `orient` command specifies the orientation of the array relative to the reference orientation, which is the orientation of the original image. The orientation angle is specified in degrees. The simulation rotates the array about its center point.

Comments may appear anywhere within the geometry command block.

B.4. Convolution Kernels

The <*convolution kernel specification*> is a sequence of records containing a row offset and a column offset, both of which are integers, and a real value between zero and one that represents the fraction of the original signal shifted by that offset. The kernel need not contain an entry in which both offsets are zero because the simulation retains counts that are not distributed to other pixels in their original pixel without an explicit entry.

The legal combinations of row and column offsets depend upon the current grid type. All pairs of integer values are legal for square and rectangular grids. ***For hexagonal grids, the difference between the offsets must be even*** because the simulation counts the intermediate columns of pixels created by shifting the even rows relative to the odd rows in

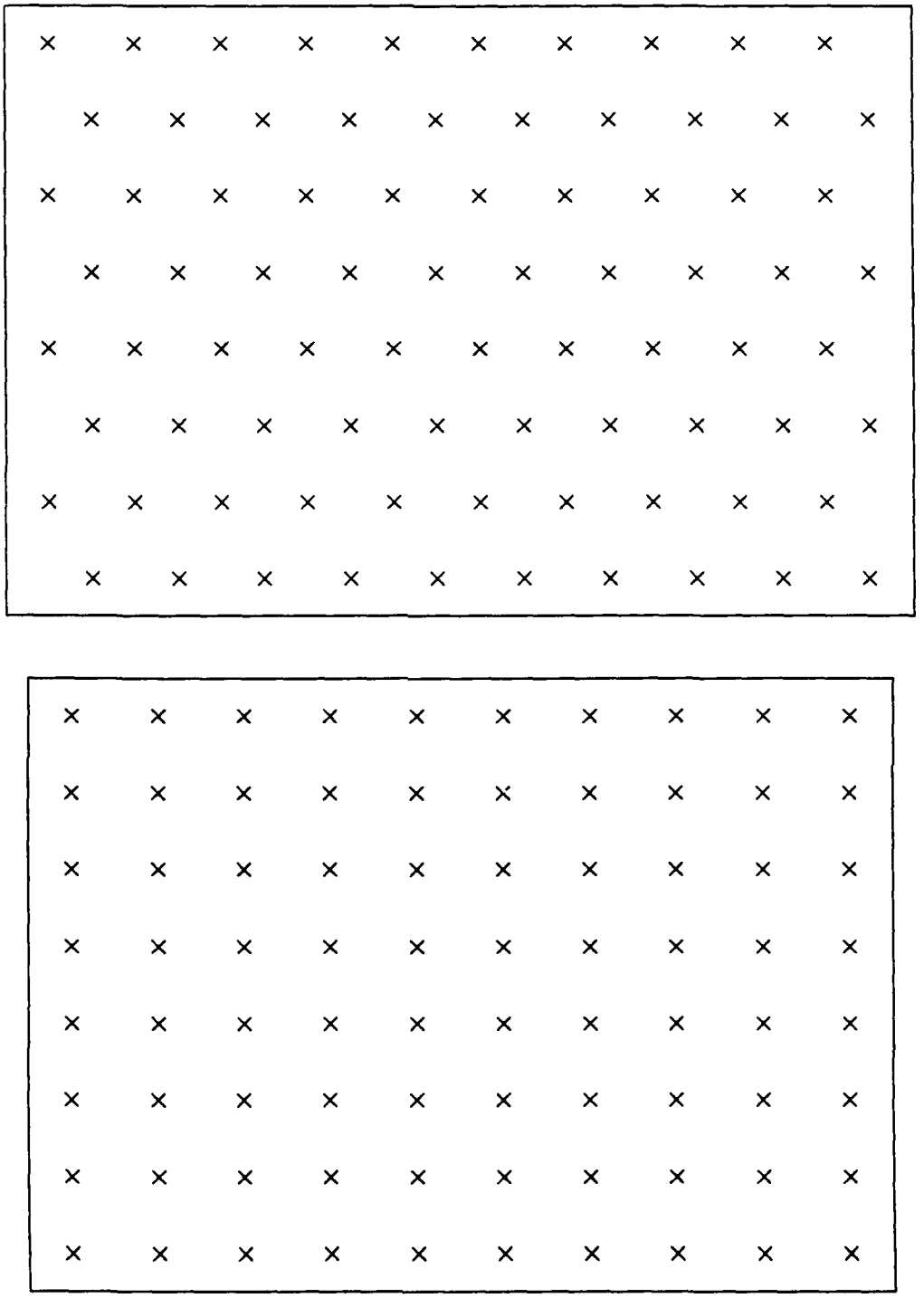


Figure B–1. Centers of pixels on hexagonal (top) and rectangular (bottom) 8×10 *grids with identical row and column separation.*

Table B–VII. Syntax for Definition of Compound Devices.

Function	Command Syntax
Arguments	`argument` *<name> <description>*
Element	`element` *<filename>*
Set Variable	`set` *<variable> <value>*
Delete Variable	`delete` *<variable>*
Comment	* *<comment text>*

computing the final offset position. Thus, proceeding clockwise from directly above a pixel in a hexagonal grid, its six nearest neighbors have (row,column) offset pairs of (-1,1), (0,2), (1,1), (1,-1), (0,-2), and (-1,-1). Similarly, *for offset grids, <modulus> must divide the difference (positive modulus) or sum (negative modulus) of the offsets* because the simulation counts the intermediate columns of pixels created by shifting the rows in each group.

B.5. Definition of Compound Devices

Definitions of compound devices are a cross between definitions of detection systems and definitions of simple devices, and thus bear some but not all characteristics of each. Table B–VII shows the commands that define compound devices.

The `element` command specifies the file defining an element of a compound device. This command is similar to the `device` command in the detection system file, but it omits the type specifier since there is only one type of element.

The remaining commands in the definitions of compound devices are identical to their counterparts in definitions of detection systems and in definitions of simple devices.

B.6. Specification of Devices of Specific Classes

Specific classes of devices, if added to the detector model, differ from general classes of devices in that the correct sequence of mathematical operations will be built into the simulation. Thus, the input file for a specific class of devices will normally contain only a sequence of parameters and identifiers to distinguish a particular device from others in the class. If appropriate, the sequence may also include names of input files corresponding to additional levels of the hierarchy.

B.7. Numerical Arguments

The detection system model accepts standard algebraic expressions for all numerical arguments for which a real (floating point) value makes sense. These expressions follow standard algebraic logic, as defined by the grammar in Table B–VIII. This grammar follows the standard precedence that evaluates expressions in parentheses first (by recursive application of the evaluation rules), followed by (unary) plus (+) and minus (-), then by multiplication (*) and division (/) from left to right, and finally by (binary) addition (+) and subtraction (-) from left to right. The detection system model also recognizes the rich set of mathematical functions shown in Table B–IX. Calls to these functions may appear

Table B–VIII. Standard Grammar for Numerical Expressions.

Entity	Permitted Expansions
Expression	*Term* *Expression*+*Term* *Expression*-*Term*
Term	*Factor* *Term***Factor* *Term*/*Factor*
Factor	*Primary* +*Primary* -*Primary*
Primary	*UnsignedConstant* (1) *Variable* (2) *FunctionReference* (3) (*Expression*)

Notes to Table B–VIII.

(1) An *UnsignedConstant* may be the symbolic name `pi`, which evaluates to the mathematical constant π, or a numerical constant written in standard floating point format consisting of a mantissa optionally followed by an exponent. The mantissa may follow one of three forms: (1) a string of decimal digits optionally followed by a decimal point, (2) a string of decimal digits followed by a decimal point and another string of decimal digits, or (3) a decimal point followed by a string of decimal digits. If the optional exponent is included, it must consist of either `e` or `E`, an optional sign (`+` or `-`), and a string of decimal digits. Strings of decimal digits must not be empty.

(2) A *Variable* is a symbolic name other than `pi`. It may consist of any sequence of alphanumeric characters, the first of which must be alphabetic, other than the symbolic name `pi` which is reserved for the mathematical constant π. The special variable `random` evaluates to a random value uniformly distributed on the unit interval. The special variable `gauss` evaluates to a random value with a normal (Gaussian) distribution of zero mean and unit standard deviation. All other names refer to predefined and user variables and physical constants.

(3) A *FunctionReference* must agree with the permitted syntax shown in Table B–IX. Each numerical argument to a function (x, y, and any additional arguments to the functions `max` and `min`) must be an *Expression* as defined in Table B–VIII.

as primaries in any expression. The evaluator uses the standard floating point (REAL) data type of the host processor for all computation.

The data function in Table B–IX interpolates an argument into a table of values contained in a separate data file. This function permits use of laboratory measurements of device characteristics. It also accommodates functional forms that are not easily represented in terms of the standard mathematical functions supported by the evaluator. For example, if a data file named s20-eta.dat contains the quantum efficiency of an S-20 photocathode as a function of wavelength in meters, the command

```
atten data('s20-eta',lambda)
```

in a device or element file represents the quantum loss at any wavelength in the domain of the data file. In this example, the simulation interpolates the tabulated data in the file to find the quantum efficiency at the system wavelength, then it draws random values for each pixel from a binomial distributions with the correct expected value and freedom parameter to simulate the noise added by the attenuation (Bernoulli selection) process.

The prototype model defines variables to the values shown in Table B–X and the metric prefixes shown in Table B–XI before it parses the files defining the detection system. The bias, atten, and gain commands also set the variable signal to the number of events in the current pixel. These names may appear as primaries in any expression without prior definition by the user. The set and delete commands cannot change or delete any of these values, but the photon and electron commands respectively set lambda and resp to the correct values. Multiplication by a unit conversion factor universally converts from the corresponding units to the internal units of the simulation. Division by a unit conversion factor likewise converts the internal units of the simulation to the indicated units.

The responsivity conversion factor, resp, converts the responsivity of a photocathode to quantum efficiency. This conversion factor allows the simulation to use values of responsivity directly in the attenuation command. If a file named s20-rsp.dat contains the responsivity of an S-20 photocathode as a function of wavelength in microns, the command

```
atten data('s20-rsp',lambda/micro)*resp
```

in a device or element file simulates the quantum loss of the photocathode. In this example, the simulation converts the wavelength from meters to microns, interpolates the tablulated data in the file s20-rsp.dat to obtain the responsivity of the detector, computes the equivalent quantum efficiency, and attenuates the signal.

The simulation does not accept expressions for arguments that require integer values, such as the dimensions of a pixel array.

B.8. References

1. S. E. Rafuse, D. Wyshogrod, N. R. Guivens, Jr., and P. D. Henshaw, "Laser Radar Sensor Simulation User's Manual," SPARTA LTR-87-011A, 26 November 1986.
2. N. R. Guivens, Jr. and P. D. Henshaw, "Target Description Language for Laser Radar System Simulation (U)," Proc. IRIS Specialty Group on Active Systems, 18–20 October 1988 (SECRET).

Table B-IX. Function Calls Permitted in Numerical Expressions.

Syntax	Description	Restrictions
`abs(`x`)`	Absolute Value	
`acos(`x`)`	Inverse Cosine	$-1 \leq x \leq 1$
`asin(`x`)`	Inverse Sine	$-1 \leq x \leq 1$
`atan(`x`)` `atan(`x,y`)`	Inverse Tangent Full Circle Inverse Tangent (1)	 $y \neq 0$ if $x = 0$
`cos(`x`)`	Cosine	(5)
`cosh(`x`)`	Hyperbolic Cosine	(5)
`data('`*file*`',x)`	Interpolate Tabulated Data in *file* (2)	
`dim(`x,y`)`	Positive Difference	
`exp(`x`)`	Exponential	(5)
`int(`x`)`	Truncate to Integer	(5)
`log(`x`)` `log(`x,y`)`	Natural Logarithm Logarithm to Base y of x	$x > 0$ $x > 0$, $y > 0$
`max(`x,y`[,`z`...])`	Maximum (Most Positive) Value	(6)
`min(`x,y`[,`z`...])`	Minimum (Most Negative) Value	(6)
`mod(`x,y`)`	Modulus y Congruence of x (3)	$y > 0$
`nint(`x`)`	Round to Nearest Integer	(5)
`pow(`x,y`)`	Raise x to y power	$x \geq 0$, $y > 0$ if $x = 0$
`sign(`x`)` `sign(`x,y`)`	Sign Function (4) Transfer of Sign (4)	
`sin(`x`)`	Sine	(5)
`sinh(`x`)`	Hyperbolic Sine	(5)
`sqrt(`x`)`	Square Root	$x \geq 0$
`tan(`x`)`	Tangent	(5)
`tanh(`x`)`	Hyperbolic Tangent	(5)

(See notes at top of next page.)

Notes to Table B–.

(1) The function call `atan(`x`,`y`)` is equivalent to `atan((`x`)/(`y`))` if $y > 0$ or to `atan((`x`)/(`y`))-sign(pi,`x`)` if $y < 0$, in keeping with the standard convention of many computer languages. If $y = 0$, this function returns 0 or π depending upon the sign of x.

(2) The function `data('`*file*`',`x`)` interpolates tabulated data in a file named *file* to obtain the function value. The data in the file must define a function by a series of data points of the form $x_i, f(x_i)$ arranged in order of increasing x_i. Each data point must appear on a separate record of the file with the two values separated either by a comma optionally followed by spaces or by one or more spaces. Blank and comment records may be placed as desired before, between, or after the data records, but comment records must not begin with numerical values. Comment text, preceeded by one of the delimiters that separate data values, may also be appended to any data record.

(3) The `mod(`x`,`y`)` function returns the congruence of the first argument on the modulus of the second argument using the standard mathematical definition of modulus. This function never returns a negative value.

(4) The function `sign(`x`)` returns `1.0` if x is positive, `-1.0` if x is negative, or `0.0` if x is zero. The function `sign(`x`,`y`)` returns `-abs(`x`)` if y is negative and `abs(`x`)` otherwise. Thus, `sign(1.0,`y`)` and `sign(`y`)` are equivalent if y is not zero.

(5) The mathematical definitions of these functions impose no restrictions on the values of their arguments, but errors may occur if the values of their arguments are not within a reasonable range determined by the host processor.

(6) The functions `min(`x`,`y`[,`z`...])` and `max(`x`,`y`[,`z`...])` require at least two arguments. These functions can have as many additional arguments as will fit into the input record. The evalutator computes these functions incrementally to minimize the chance of stack overflow.

Table B–I. Predefined Variables.

Name	Value	Units
`lambda`	System Wavelength (λ)	Meters
`resp`	Responsivity Conversion Factor ($hc/\lambda e$)	Watts/Amp
`hc`	Photon Wavelength-Energy Product (hc)	Joules/Meter-Count
`elec`	Charge of Electron (e)	Coulombs/Count

Table B–XI. Predefined Unit Conversion Factors.

Name	Value
`milli`	10^{-3}
`micro`	10^{-6}
`nano`	10^{-9}
`pico`	10^{-12}

Name	Value
`kilo`	10^{3}
`mega`	10^{6}
`giga`	10^{9}
`tera`	10^{12}

Appendix C. The Prototype Code

SPARTA implemented a prototype code for the detection system model to demonstrate the concept of the model and to provide a test code for refinement of the model before its incorporation into SPARTA's optical sensor simulation, SENSORSIM, and the simulation module of the Defense Laser/Target Signatures (DELTAS) code. Written entirely in FORTRAN 77 with minor extensions described below for compatibility with the SENSORSIM and DELTAS codes, this prototype code follows the same basic structure as the detection system model. The code's main routine corresponds corresponds to the overall detection system at the top level of the model's hierarchy, with separate subroutines for simple and compound devices, elements, and mathematical operations at each subordinate level. Numerous utility routines perform standard functions such as parsing input and generating random numbers with various distributions. Two additional modules manage image buffers and variables defined by the user.

C.1. Overall Structure

The structure of the prototype code follows the basic hierarchical structure of the detection system model. The main program simulates the overall detection system, calling subroutines that simulate each class of device. The subroutines `SIMPLE` and `COMPND` respectively simulate simple and compound devices. The subroutine `COMPND` in turn calls the subroutine `ELEMNT` to simulate each element of a compound device. In the prototype code, the subroutine `ELEMNT` is actually a second entry point in subroutine `SIMPLE`, as this allows both routines to share a large block of common code.

The core of the simulation's main program is a nested "block `IF`" logic tree in a loop over the records of the detection system files. The subroutines `SIMPLE` and `COMPND` contain similar logic trees that loop over the records of their respective input files. These logic trees call the parsing routines `PARSE`, `NUMARG`, and `INTVAL` as necessary to parse and interpret each command, executing each command once it is interpreted. The main program also contains code to initialize the model and to save the final image.

The parsing routines `PARSE`, `NUMARG`, and `INTVAL` provide full parsing capabilities for all command files. The subroutine `PARSE` returns the next token in the command line as a text string. The function `NUMARG` is a sophisticated numerical expression evaluator that computes the value of floating point arguments to each command, calling a user variable handler, described below, to obtain the value of each variable or constant other that `pi`, `random`, and `gauss` that appears in an expression. The function `INTVAL` reads integer values from the record, but it does not provide any support for integer expressions beyond a simple value.

The subroutine `SIMPLE` calls a set of subroutines that execute mathematical operations according the commands in the definition of each simple device or element. Table C-I shows the subroutines that execute mathematical operations in the prototype code. To add another mathematical operation to the simulation, one need only define a command corresponding to the operation, write and test a subroutine to execute the operation, and add a few statements to the logic tree in `SIMPLE` to parse the command and call the

Table C–I. Mathematical Operations in the Prototype Model.

Operation	Routine
Losses (Bernoulli Selection)	ATTEN
Amplification	AMPLFY
Fixed (D. C.) Bias	BIAS
Change Pixel Geometry	PXLMAP
Spread Signal to Other Pixels	SPREAD

Table C–II. Pixel Geometry Routines.

Operation	Routine
Area of Intersection of Two Circles	CCAREA
Area of Polygon	PGAREA
Extreme Limits of Polygon	PGEXTR
Area of Intersection of Polygon and Circle	PGINTC
Overlap of Two Polygons	PGOVRL
Vertices of Rectangle	PGRECT
Vertices of Regular Polygon	PGREGN
Rotate Vertices of Polygon	PGTURN

subroutine. The main program also calls the subroutine AMPLFY to simulate shot noise as part of its initialization process.

The subroutines that execute mathematical operations call several routines that generate random values with various distributions. For signals expected to exceed forty (40) counts, the mathematical routines approximate the distributions of the binomical family by rounding values drawn from Gaussian distributions with the same mean and standard deviation to the nearest integer value. The subroutine PXLMAP also calls the geometry routines shown in Table C–II to determine the overlapping area of the old and new pixels.

The simulation's image buffer manager, described below, maintains the image pixel array and associated parameters. The main program calls the image buffer manager to load the focal plane image and to write images to files. All other routines call the appropriate functions of the image buffer manager to obtain or change image values.

C.2. User Variable Handler

The user variable handler is a single module with the functions shown in Table C–III. These functions return the LOGICAL values .TRUE. if successful and .FALSE. on error.

The variable handler provides two functions to manage memory for storage of user variables. The function USRVAR either allocates memory for a specified number of variables or verifies that the memory previously allocated can accommodate a specified number of

Table C–III. User Variable Handler Functions.

Operation	Routine
Allocate/Verify Storage	USRVAR
Set Value of Variable	SETVAR
Return Value of Variable	GETVAR
Add Value to Variable	ADDVAR
Multiply Variable by Value	MLTVAR
Delete Variable	DELVAR
Release Variable Storage	DONVAR

additional variables. The function DONVAR deallocates this memory, thus deleting all user variables.

The remaining functions of the variable handler operate on individual variables. The function SETVAR sets the value of a variable, creating the variable if it does not already exist. The function GETVAR returns the current value of a variable to the calling routine. The function ADDVAR adds a value to the current value of a variable, recording the result as the new value of the same variable. The function MLTVAR similarly multiplies the current value of a variable by a specified value. Finally, the function DELVAR deletes a variable, allowing another variable to take its place.

The prototype code allocates storage for sixty-four (64) variables. Fourteen (14) of these variables are reserved for the predefined physical constants and unit conversion factors. The remaining fifty variables are available for the definition of the detection system and its components.

C.3. Image Buffer Manager

The image buffer manager is a module containing the functions shown in Table C–IV. These functions manage memory for image buffers, store and retrieve images, manipulate image pixel values, and manage supplemental specifications associated with each buffer. The functions PIXEL and SPEC return the values that they retrieve as the value of the function. All other functions return the LOGICAL values .TRUE. if successful and .FALSE. on error.

The image buffer manager maintains nain and auxilliary image buffers. When only one buffer is active, it is always the main image buffer. When both buffers are active, the buffer created most recently is always the main buffer.

The function NEWBUF creates a new image buffer with dimensions specified by the calling routine. The function IMGBUF creates a new buffer and loads an image into it from a disk file. The function WRTBUF writes a copy of an image to a disk file. The function DELBUF deletes an active buffer. The function DONBUF deletes all active buffers. The function BUFDIM tests the current status of a buffer and returns the buffer dimensions if the buffer is active.

Several functions in the buffer manager manipulate image pixel values. The functions SETPIX, ADDPIX, and MLTPIX respectively set a single pixel to a value, add a value to a

Table C–IV. Image Buffer Manager Functions.

Function	Name
Reserve New Image Buffer	NEWBUF
Load Image to New Buffer	IMGBUF
Write Buffer to Image File	WRTBUF
Return Buffer Dimensions	BUFDIM
Initialize Buffer	SETBUF
Add Value to Buffer	ADDBUF
Multiply Buffer by Value	MLTBUF
Set Pixel Value	SETPIX
Add Value to Pixel	ADDPIX
Multiply Pixel by Value	MLTPIX
Return Pixel Value	GETPIX
as Function Value	PIXEL
Duplicate Specifications	DUPSPC
Set New Specifications	NEWSPC
Return All Specifications	ALLSPC
Set Single Specification	SETSPC
Return Single Specification	GETSPC
as Function Value	SPEC
Delete Buffer	DELBUF
Delete All Buffers	DONBUF

single pixel, and multiply a single pixel by a value. The functions SETBUF, ADDBUF, and MLTBUF perform the same respective operations uniformly on each pixel of a buffer. The functions GETPIX and PIXEL return the value of a pixel to the calling routine.

The buffer manager also maintains supplemental specifications for each image buffer. These supplemental specifications are real values that are stored in the image file. The function DUPSPC copies the specifications from the auxilliary buffer to the main buffer. The function SETSPC and sets a single specification. The functions GETSPC and SPEC return a single specification to the calling routine. The functions NEWSPC and ALLSPC respectively set and return all specifications. The function NEWSPC also reallocates the specification array, permitting changes in its size. These supplemental specifications are written to each image file along with the image dimensions and pixel values.

C.4. Programming Language

The use of FORTRAN '77 for the SENSORSIM and DELTAS simulations, to which the new detection system model may be added in the future, made FORTRAN the language of choice for the prototype source code as well. The X3J3 (FORTRAN) subcommittee of the American National Standards Institute (ANSI) has developed a new standard for the FORTRAN programming language, commonly known as FORTRAN '90, which is a proper superset of FORTRAN '77. At the start of this project, final approval of the FORTRAN '90 standard was thought to be imminent, but it is now pending resolution of several peripheral issues [1].

The source code for the prototype model conforms as far as possible to the FORTRAN '77 standard [2], thus ensuring maximum possible compliance with the anticipated FORTRAN '90 standard. The prototype source code does contain a few extensions to perform operations that are not possible within the FORTRAN '77 standard, but it can run on any processor conforming to the ANSI FORTRAN '77 standard with minor modifications described below. The prototype code compiles correctly with version 5.0 or later of the Microsoft FORTRAN compiler on IBM or compatible personal computers.

The prototype code has a simple "prompt and response" interface similar to that of SENSORSIM. Two of the extensions enhance this interface. First, several console `WRITE` statements in the main program contain backslash (\) characters in their format strings. This Microsoft extension causes the console `READ` statements following the affected `WRITE` statements to display the user's input on the same line as the prompt rather than on the following line. Second, the main program calls the subroutine `CLRSCR` to clear the screen. This subroutine clears the screen by writing an ANSI (VT-100) terminal control sequence that is not defined in the FORTRAN '77 standard. These extensions may be replaced with equivalent extensions of another processor or removed completely without affecting the results generated by the program.

The prototype code uses SENSORSIM's image input and output routines to store and retrieve image files. These routines open image files with Microsoft's `FORM='BINARY'` extension in their `OPEN` statements to provide compatibility with the input and output system of the C programming language. SENSORSIM's image input and output routines also compress images in a lossless run length encoded format that adds only four bytes to files containing uncompressible images. These routines can be replaced with image input and output routines for another environment by changing a few subroutine calls in the image buffer manager.

Several routines in the prototype code use dynamic memory allocation to avoid potentially unacceptable compromises between excessive memory requirements and limitations on capacity. Microsoft's implementation of dynamic memory allocation is very close to the proposed FORTRAN '90 standard, differing only in the syntax for declaring the `ALLOCATABLE` attribute of the affected arrays. Removal of this extension entails changing the declarations of the affected arrays to specify fixed dimensions and replacing the `ALLOCATE` and `DEALLOCATE` statements, calls to the `ALLOCATED()` intrinsic function, and associated error handling code with code to check the adequacy of the fixed dimensions.

C.5. References

1. ANSI Staff, Phone Conversations with N. R. Guivens, Jr. of SPARTA, Inc., August 1991–August 1992.
2. *American National Standard Programming Language FORTRAN*, ANSI X3.9-1978, American National Standards Institute, New York, 1978.